**Compiler Design Lab**

**Practical-6**

**Name: Yash Patni**

**Roll: 71**

**Batch: A4**

**Topic: Code Optimization**

**Aim: Write a code to implement Local optimization techniques until no further optimization is possible for the given three address code.**

**Code:**

def optimize\_TAC(tac):

    optimized\_tac = tac

    def copy\_propagation(tac):

        var\_map = {}

        optimized\_tac = []

        for line in tac:

            tokens = line.split()

            if len(tokens) == 3 and tokens[1] == "=":

                if not tokens[2].isdigit():

                    var\_map[tokens[0]] = tokens[2]

                optimized\_tac.append(line)

            else:

                new\_line = " ".join([var\_map.get(token, token) for token in tokens])

                optimized\_tac.append(new\_line)

        return optimized\_tac

    def constant\_propagation(tac):

        constants = {}

        optimized\_tac = []

        for line in tac:

            tokens = line.split()

            if len(tokens) == 3 and tokens[1] == "=" and tokens[2].isdigit():

                constants[tokens[0]] = tokens[2]

                optimized\_tac.append(line)

            else:

                new\_line = " ".join([constants.get(token, token) for token in tokens])

                optimized\_tac.append(new\_line)

        return optimized\_tac

    def constant\_folding(tac):

        optimized\_tac = []

        for line in tac:

            tokens = line.split()

            if len(tokens) == 5:

                op1, operator, op2 = tokens[2], tokens[3], tokens[4]

                if op1.isdigit() and op2.isdigit():

                    result = eval(f"{op1} {operator} {op2}")

                    new\_line = f"{tokens[0]} = {result}"

                else:

                    new\_line = line

                optimized\_tac.append(new\_line)

            else:

                optimized\_tac.append(line)

        return optimized\_tac

    def common\_subexpression\_elimination(tac):

        subexpr\_map = {}

        optimized\_tac = []

        for line in tac:

            tokens = line.split()

            if len(tokens) == 5:

                subexpr = " ".join(tokens[2:])

                if subexpr in subexpr\_map:

                    new\_line = f"{tokens[0]} = {subexpr\_map[subexpr]}"

                else:

                    subexpr\_map[subexpr] = tokens[0]

                    new\_line = line

                optimized\_tac.append(new\_line)

            else:

                optimized\_tac.append(line)

        return optimized\_tac

    def dead\_code\_elimination(tac):

        used\_variables = set()

        assigned\_variables = set()

        optimized\_tac = []

        for line in tac:

            tokens = line.split()

            assigned\_variables.add(tokens[0])

            if len(tokens) == 5:

                used\_variables.add(tokens[0])

                used\_variables.add(tokens[2])

                used\_variables.add(tokens[4])

        for line in tac:

            variable = line.split()[0]

            if variable in used\_variables or variable not in assigned\_variables:

                optimized\_tac.append(line)

        return optimized\_tac

    prev\_tac = []

    print("Original TAC:")

    print("\n".join(tac))

    print("\nOptimizing with copy propagation:")

    optimized\_tac = copy\_propagation(optimized\_tac)

    print("\n".join(optimized\_tac))

    print("\nOptimizing with constant propagation:")

    optimized\_tac = constant\_propagation(optimized\_tac)

    print("\n".join(optimized\_tac))

    print("\nOptimizing with constant folding:")

    optimized\_tac = constant\_folding(optimized\_tac)

    print("\n".join(optimized\_tac))

    print("\nOptimizing with common\_subexpression\_elimination:")

    optimized\_tac = common\_subexpression\_elimination(optimized\_tac)

    print("\n".join(optimized\_tac))

    print("\nEliminating dead code:")

    optimized\_tac = dead\_code\_elimination(optimized\_tac)

    print("\n".join(optimized\_tac))

    while prev\_tac != optimized\_tac:

        prev\_tac = optimized\_tac

        optimized\_tac = copy\_propagation(optimized\_tac)

        optimized\_tac = constant\_propagation(optimized\_tac)

        optimized\_tac = constant\_folding(optimized\_tac)

        optimized\_tac = common\_subexpression\_elimination(optimized\_tac)

        optimized\_tac = dead\_code\_elimination(optimized\_tac)

    return optimized\_tac

tac = [

    "a = 2",

    "b = x \* x",

    "c = x",

    "d = a + 5",

    "e = b + c",

    "f = c \* c",

    "g = d + e",

    "h = e \* f"

]

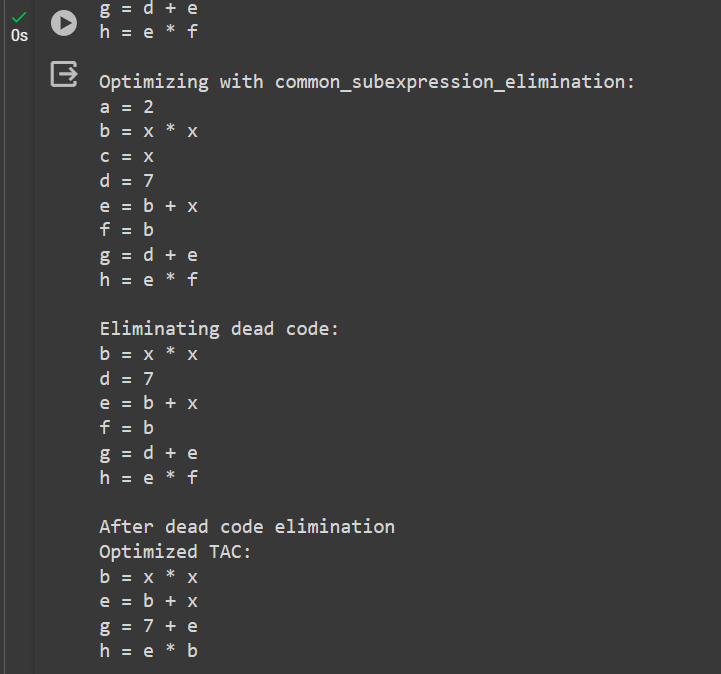
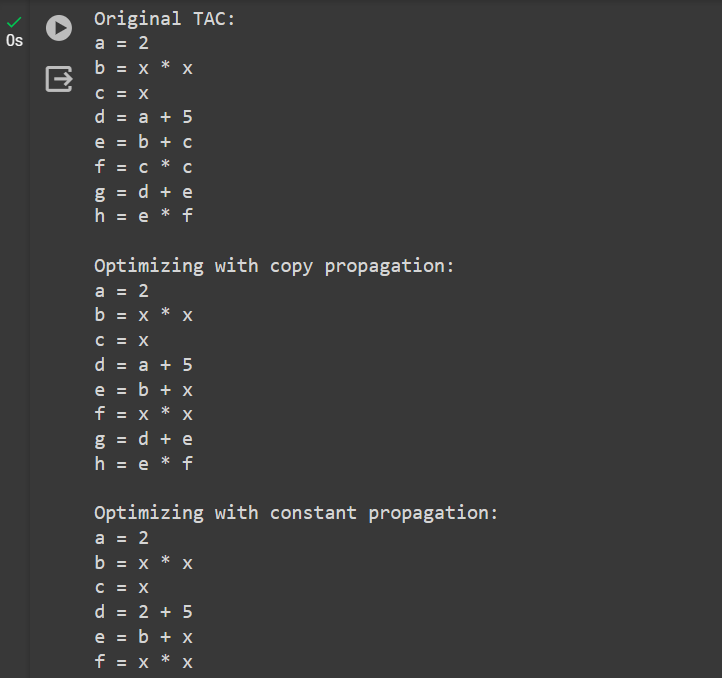
optimized\_tac = optimize\_TAC(tac)

print("\nAfter dead code elimination")

print("Optimized TAC:")

print("\n".join(optimized\_tac))

**Output:**

****

**Conclusion:** In this practical we successfully implemented local optimization techniques and optimized the given 3 address code.